Threatlyzer: Advanced Security System With Proactive Detection

Piyush Kumar Singh
SCAI
VIT Bhopal University
Bhopal, India
piyush.24bai10094@vitbhopal.ac.in

Ansari Murtaza Muzammil Akhtar
SCAI
VIT Bhopal University
Bhopal, India
ansari.24bai10109@vitbhopal.ac.in

Muskan Sukhija
SCAI
VIT Bhopal University
Bhopal, India
muskan.24bai10927@vitbhopal.ac.in

Vanshika Agrawal
SCAI
VIT Bhopal University
Bhopal, India
vanshika.24bai10092@vitbhopal.ac.in

Shivangee Gupta
SCAI
VIT Bhopal University
Bhopal, India
shivangee.24bai10056@vitbhopal.ac.in

Abhradwip Lala
SCAI
VIT Bhopal University
Bhopal, India
abhradwip.24bai10657@vitbhopal.ac.in

***Abstract*- This project presents the Threatlyzer app, an advanced security system designed or the proactive detection and isolated content analysis of both known and unknown malware variants. The system utilizes a dual-path detection mechanism, combining the speed of traditional signature-based detection with the adaptability of Machine Learning (ML) models, specifically Decision Tree and Support Vector Machine (SVM) classifiers. Feature extraction is performed on file metadata, including TLSH checksums and network activity indicators, to accurately classify threats. Upon detection, the malicious file is contained using a sandboxing technique and its content is extracted and analyzed in a secure, isolated window. A Large Language Model (LLM) and a Bigdata-Hadoop framework are then used to generate a detailed, human-readable summary and analysis report of the file's potential actions and effects. The system achieved a classification accuracy of 91% and a malware precision of 94% on the test dataset. This approach not only provides a valuable layer of proactive defense against evolving threats but also enables safe, deep inspection of otherwise inaccessible data within infected files.**

***Keywords*-** *Malware detection, Machine Learning, Sandboxing, LLM, Big Data, Signature-Based Detection, Decision Tree, SVM, TLSH, Proactive Security.*

I. INTRODUCTION

An App for the Malware Detection and Extraction of File Content in Isolated Window:

The field of cybersecurity is currently facing an unprecedented challenge due to the rapid evolution and proliferation of malicious software. Approximately 190,000 new malware variants are created globally every second, leading to increasingly sophisticated cyberattacks that can compromise sensitive systems, including military networks, and render confidential internal data inaccessible. Traditional malware detection approaches, such as signature-based detection, are becoming increasingly inadequate against modern, polymorphic and zero-day threats because they are fundamentally reactive—they can only identify a threat after its signature has been documented and added to a database.[3]

To address this challenge, we propose the development of integrated signature- based and machine learning model malware detection, isolated window technique for the data extraction and read only file content availability.

A. The Problem:

The necessity to safely access and analyse the content of a file that has already been infected and rendered inaccessible. Building an automated app that can detect existing and new malware variant, extracting the file content in an isolated window, storing the tokens for the future detection.

B. The Solution:

Our app Threatlyzer provide the unique way to handle the new malware variants following robust approach traditional signature-based detection and machine learning models it leads to the analysis of the new malware variant, with the LLM and Bigdata-Hadoop provides the summary and detailed overview of the file in an isolated system, following Sandboxing technique.

C. The Objective:

This project introduces **Threatlyzer**, a robust and automated application designed to address these challenges. Threatlyzer provides a comprehensive approach by integrating a **dual-detection methodology** with a secure, isolated analysis environment.

The system's main objectives are to:

- **Detect** existing and new malware variants using a combination of **traditional signature-based** and **Machine Learning (ML)** models.
- **Isolate** the detected malicious file using a **Sandboxing technique** to prevent any execution or harm to the host system.
- **Extract** the file content and generate a comprehensive **summary** and detailed overview, using a **Large Language Model (LLM)** and a **Bigdata-Hadoop** framework, providing security teams access to otherwise inaccessible information.
- **Create tokenized fingerprints** of newly identified malware for future detection, enhancing continuous, adaptive security.

D. Paper Organization:

The rest of this paper is structured as follows: Section II reviews related work investigation in detection methodologies, Section III details the requirement artifacts, Section IV details the design methodology and its novelty, Section V details the technical implementation & analysis, Section VI details the project outcome and applicability, Section VII details the conclusions and recommendation.

II. RELATED WORK

The field of cybersecurity is a dynamic landscape where the sophistication of malicious software evolves rapidly. Traditional malware detection methods, such as those relying on static signatures, are becoming increasingly inadequate against modern, polymorphic viruses designed to evade detection. This chapter investigates the current state of malware detection, examining both historical and contemporary approaches to identify their strengths, weaknesses, and the motivation for this project's methodology.

A. Core Area of the Project:

The core area of this project is the classification of malware using traditional signature-based and machine learning models, opening malware variant file in an isolated window for better security. This approach provides a lightweight and efficient alternative tool. It focuses on extraction of the malware file content and the summary of the malware file providing with a new way to look inside the malware files.

B. Existing Approaches/Methods:

- **Signature-Based Detection:** Signature-based detection is the most common and oldest method for identifying malware. It works by creating a unique digital fingerprint, or signature, for known malicious files. Antivirus software then compares the signature of a new file with its database of known malware signatures. If a match is found, the file is flagged as malicious. This method is effective for identifying previously seen malware variants.
- **Heuristic-Based Detection:** Heuristic-based detection goes beyond simple signatures by using a set of rules or heuristics to identify suspicious behavior or code patterns. Instead of looking for an exact signature, this method analyses a file for characteristics commonly found in malware, such as self-modification, attempts to access the system registry, or a lack of valid file headers. This allows it to detect some new or previously unknown malware.
- **Machine Learning-Based Detection:** This modern approach uses machine learning algorithms to analyse a vast number of features extracted from files. These features can include metadata, code structure, or behavioral patterns. By training a model on a dataset of both benign and malicious files, the algorithm learns to identify complex, non-obvious patterns that can distinguish between the two classes. This project specifically employs a Decision Tree model for its interpretability and a SVM for its effectiveness in high-dimensional feature spaces.[1],[2],[3]

C. Pros and Cons of the Stated Approaches/Methods:

Approach	Advantages	Disadvantages
Signature-Based	Highly accurate for known malware. Fast and has a low false-positive rate.	Ineffective against zero-day and polymorphic malware. Requires a constantly updated signature database.
Heuristic-Based	Can detect some unknown malware variants. Does not require a massive signature database.	Can result in a high number of false positives. Rules can be evaded by sophisticated attackers.
Machine Learning-Based	Highly adaptable to new and evolving threats. Capable of detecting previously unknown (zero-day) malware. Can handle large datasets of complex features.	Requires a large, high-quality, and well-labeled dataset. Training can be computationally expensive. The model can sometimes act as a "black box," making it hard to interpret its decisions.

D. Issues/Observations from Investigation:

The investigation into existing malware detection methods revealed that traditional approaches have significant limitations when confronted with the dynamic and ever-changing nature of modern malware. The core issue is the reactive nature of signature-based systems, which can only identify a threat after its signature has been documented and added to a database. Heuristic-based systems, while more proactive, often struggle with a high rate of false positives. The key observation is that a machine learning-based approach, trained on carefully selected metadata, offers a powerful and efficient solution to these problems. It provides a crucial layer of proactive defense by learning to recognize patterns rather than relying on an exact match, which is the foundational principle behind this project.[2],[3]

III. REQUIREMENT ARTIFACTS

This section details all the requirements software, hardware, and specific functionalities necessary for implementing the advanced malware detection application. The artifact specification ensures a clear understanding of what is needed for smooth development and deployment of the

system, leveraging a tech stack tailored for Python-based desktop applications.

A. Hardware and Software Requirements:

This section lists and describes the key hardware and software resources essential for the project's successful execution.

1. Hardware Requirements:

A standard desktop or laptop with at least Intel or AMD processor (minimum dual-core, 2.0 GHz or higher), 4 GB RAM or above, 500 MB free disk space for application and dataset storage, Windows/Linux operating system (system compatibility for Python and Tkinter).

2. Software Requirements:

Programming Language- Python 3.8+ (for all development tasks and integration), Graphical User Interface Toolkit- Tkinter (standard module in Python for GUI design), Operating System Interaction: Python's built-in O.S. module for management of files and folders, hashing utility- hashlib (standard library, uses SHA256 for secure file fingerprinting), text editor/IDE- Any compatible editor, e.g., VS Code, PyCharm, or IDLE, optional- Internet connectivity (for updates and dataset acquisition).

B. Specific Project Requirements:

1. Data Requirements:

- The system must handle multiple file types (executables, docs, etc.) to perform malware scanning and content extraction.
- A dataset containing both malicious and benign file fingerprints/signatures, preferably in JSON or CSV formats.
- Storage for tokenized representations of newly detected malware files for later reference.

2. Functions Requirement:

- Secure uploading and handling of files through the GUI.
- Accurate file fingerprinting using SHA256 hashing via hashlib.
- Real-time file deletion or management using O.S. remove.
- Automated and manual file classification via signature-based and machine learning models.
- Safe extraction and display of file contents in an isolated window using Tkinter.
- User-driven actions for retaining or deleting files upon detection.

3. Performance and Security Requirement:

- All hashing and file operations must be performed efficiently, within a reasonable time (preferably under 1 second for standard files).
- The isolation (sandboxing) of scanned files must prevent any execution or spread of malware in the host system.
- Strict file access control and logging for user actions on each scanned item

4. Look and Feel Requirements:

- Clean, intuitive desktop application interface built using Tkinter.
- Native widgets (buttons, labels, windows, dialogs) ensuring familiarity and ease of navigation.
- Responsive design and prompt feedback for user operations (scan, isolate, summary, delete).

C. Underlying Technologies:

- **Programming Language:** Python- Chosen for its simplicity, reliability, and extensive standard libraries that facilitate seamless rapid development.
- **GUI: Tkinter-** Used for its native look and robust support in Python for creating desktop applications and window management.
- **Hashing Algorithm:** Hashlib (SHA256)-Provides cryptographically secure hashing for file fingerprinting, ensuring reliable detection and identification of malware variants.
- **ML Models:** Decision Tree and SVM – Used for the extraction of the features, training on the JSON files and prediction for the Malware variants.
- Sandboxing: Python sandbox libraries are used to build an isolated window which separates the system processes with it.
- **OS Interaction:** OS Module-Enables essential interactions with file paths and actual deletion of files from the system, forming the backbone of system-level operations.

IV. DESIGN METHODOLOGY AND ITS NOVELTY

A. Methodology and goal:

The primary goal of the Threatlyzer application is to provide a comprehensive and automated solution for the detection and analysis of both known and unknown malware variants, displaying the file content in an isolated system. The core methodology follows a linear pipeline design that integrates both traditional and modern security analysis techniques. This approach aims to provide a robust defense by combining the speed of signature-based detection with the adaptability of machine learning.

The overall methodology is defined by the following sequential steps:

1. **File Ingestion:** The user selects a file or a folder to be scanned.
2. **Feature Extraction:** Key metadata (e.g., file size, network activity indicators, and TLSH checksums) is programmatically extracted from the file's JSON analysis report.

tlsh	name	size	type
DDA30242B51879F6C1C199330662CD202C7D3C3478F9CE5B58F6BA29D9B7C6AB184E93	VirusSign Sample	97796	x-msdownload
2124C003B4D2C8F6C19746B50CA6ABEDF27AD9700B2385C3B3945E7E2C7A1D0563A356	Virus	227840	application
88e76e11ffd91f5352600cc6c501ea256f176f62f4f81342ee9208779c0e80b9	Tutorial 1 chy	77104	pdf
454e5a32b53c8c67982417568e2857fc200968be1ae710ed968eb1d95b0b94e3	LSBSetup	3812528	exe

3. **Dual-Path Detection:** The extracted data is simultaneously analyzed by a Signature Based Detector (for known threats) and a Machine Learning Classifier (for novel threats).
4. **Sandboxed Analysis:** If a threat is detected by either method, the file is processed in an isolated environment using a sandboxing technique.[4], [5]
5. **Content Extraction and Summary:** Within the sandbox, the file's content is extracted, and a summary and detailed analysis are generated using LLM and a Bigdata-Hadoop framework. [4], [5]
6. **Token Creation:** New signatures or tokens are created for newly identified malware variants, which are then added to the system's database for future detection.

B. Algorithm:

Step 1: **START:** Receive input file F.

Step 2: Extract Metadata M from F (Size, File Type, Network Indicators).

Step 3: Calculate **TLSH Checksum** H_{TLSH} from F.

Step 4: Form Feature Vector V={M,H_{TLSH}}.

Step 5: Check if Hash(*F*) exists in the Signature Database D_{SIG}.

Step 6: If Hash(*F*)∈D_{SIG}, then Status←Malicious (Known). **GOTO Step 11.**

Step 7: Apply trained Machine Learning Classifier *C* (Decision Tree and SVM) to V.

Step 8: P←C(V), where P is the probability of being malicious.

Step 9: If P≥Threshold, then Status←Malicious (Zero-Day): **GOTO Step 11.**

Step 10: If Status←Benign: **END.**

Step 11: If Status←Malicious:

Step 12: Isolate: Route *F* to the secure Sandboxing Environment *S* (isolated process).

Step 13: Content Extraction: Within *S*, safely extract raw file content *C*.

Step 14: Process content *C* using the Bigdata-Hadoop framework to handle scale and structure.

Step 15: Feed processed content to the Large Language Model (LLM) *L*.

Step 16: *L* generates a detailed summary and threat report *Report*.

Step 17: If Status←Malicious (Zero-Day):

Step 18: **Tokenize:** Generate new, unique signature Token for *F*.

Step 19: **Update Database:** Add Token and Hash(*F*) to $D_{SIG.}$

Step 20: Display the Report within the secure, isolated UI.

Step 21: **END.**

C. Functional Modules Design and Analysis:

The Threatlyzer application is designed with a modular approach, where each function is handled by a distinct module. This design promotes maintainability, scalability, and clarity.

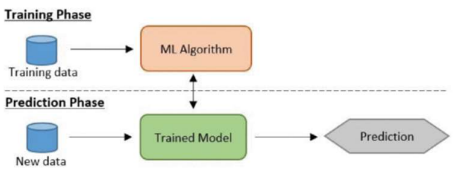
The key modules are:

File Ingestion Module: This module handles user interaction, allowing the selection of a single file or a folder. It manages the queue of files to be scanned, ensuring that the entire directory is processed efficiently.

Feature Extraction Module: This component is responsible for parsing the input file's JSON metadata. It identifies and extracts the required features, transforming raw data into a structured format that can be used by the machine learning model.

Detection Engine: This is the core of the system, comprising two sub-modules:

1. **Signature-Based Detector:** Compares the file's hash against a pre-existing database of known malware signatures.
2. **Machine Learning Classifier:** Utilizes the trained Decision Tree and SVM models to classify the file based on its extracted features.



Model division and working flow.[6]

Mathematical Equation for the Model:

$$Gini(D) = 1 - \sum_{i=1}^{m} (pi)^2$$

Where:

- D is the current set of data (e.g., all files at a node).
- m is the number of classes (malware, benign).
- pi is the probability of an element being classified to class i (the proportion of files belonging to class i).

The feature that results in the greatest **Information Gain** (the largest reduction in Gini Impurity after the split) is chosen.

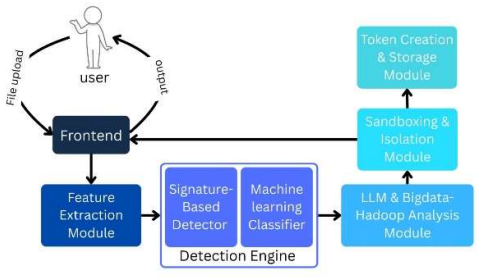
Sandboxing & Isolation Module: This module provides a secure environment for the execution and analysis of potentially malicious files. It ensures that the file's actions are contained, preventing any harm to the host system. [4], [5]

LLM & Bigdata-Hadoop Analysis Module: This advanced module is triggered when a threat is detected. It processes the extracted file content using an LLM to generate a human-readable summary and detailed report. The use of a Bigdata-Hadoop framework allows for the efficient processing of large files and data sets. [4], [5]

Token Creation & Storage Module: This module is critical for proactive defense. It takes the key characteristics of a newly identified malware variant and generates a new token or signature, which is then securely stored for future use.

D. Software Architectural Designs:

The software architecture of the Threatlyzer application is based on a **sequential pipeline model**, where data flows through a series of interconnected stages. This design ensures that each operation is completed before the next one begins, creating a predictable and logical flow.



Software Architectural Design Diagram.

The architecture is a single-process, desktop application built using Python's tkinter for the UI and standard libraries for the core logic. While the main process is sequential, the tkinter UI runs in a separate thread from the core scanning logic to prevent the application from freezing. The LLM & Bigdata-Hadoop Analysis Module can be considered an external, microservice-like component that is invoked on demand, maintaining the system's lightweight nature.

E. Subsystem Services:

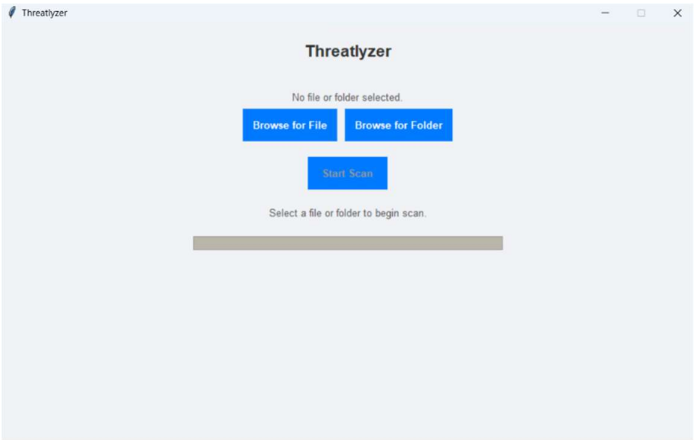
The modular design allows for the clear definition of key subsystem services:

- **Scanning Service:** Orchestrates the entire scan process, including file queuing, feature extraction, and calling the detection engine.
- **Classification Service:** This service encapsulates the Decision Tree and SVM models, providing a single, clean interface for predicting a file's threat level. It takes a feature vector as input and returns a classification (malicious or benign).
- **Threat Reporting Service:** This service manages the interaction with the LLM/Bigdata-Hadoop component. It sends the file content to the service and receives the detailed analysis report, which is then formatted for display.
- **File Management Service:** Handles file-system operations, such as deleting or moving detected malicious files to a quarantine location.

F. User Interface Designs:

The user interface for Threatlyzer is designed to be simple, intuitive, and efficient. It is implemented using Python's tkinter library.

The main window includes:

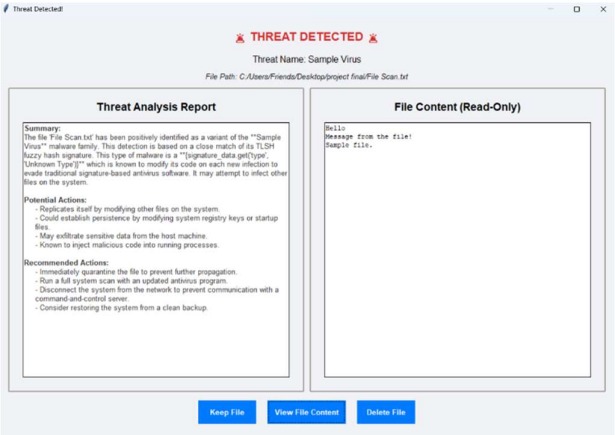


Main Window of the project app

- **File/Folder Selection:** Two prominent buttons allow the user to browse and select a file or a folder for scanning.
- **Status Display:** A status label provides real-time feedback on the scanning process, indicating the current file being scanned and the overall progress.

- **Progress Bar:** A progress bar visually represents the scan's completion percentage, giving the user a clear sense of the remaining time.
- **Scan Button:** A central button to initiate the scanning process, which is disabled when no file is selected.

Upon threat detection, a separate **Isolated Window** appears. This window is a key part of the sandboxing technique and provides:



Isolated Window comprising Threat Analysis Report and Read Only File Content.

- **Clear Threat Alert:** A bold title, such as "Threat Detected!", immediately alerts the user.
- **Threat Details:** Information about the detected file's name and path.
- **Action Buttons:** A set of clear actions for the user, such as "Delete File", "Keep File" or “View File Content”.
- **Detailed LLM Report:** A scrollable text area displays the comprehensive threat analysis report generated by the LLM, including a summary, potential actions, and recommendations.
- **File Content View:** A Read-Only file content is available to see the content of threated file.

V. TECHNICAL IMPLEMENTATION & ANALYSIS

A. *Technical Coding and Code Solutions:*

The core of the Threatlyzer application is written in **Python**. The technical implementation is structured to be modular and easy to understand. The key components include:

- **Feature Extraction:** This module uses Python's json and os libraries to iterate through files in a specified directory. For each JSON file, a function extract_features is called to parse the data and extract relevant numerical and categorical features (e.g., file_size_kb, has_network_activity). This process transforms raw data into a format suitable for machine learning.
- **Data Preprocessing and Training:** The sklearn library is used to handle data processing. The TfidfVectorizer is applied to text-based features to convert them into numerical representations. The preprocessed data is then split into training and testing sets using train_test_split.
- **Model Implementation:** The **Decision Tree** classifier is imported from sklearn.tree and the **Support Vector Machine (SVM)** classifier from sklearn.svm. The models are trained on the preprocessed data using the fit method.
- **Prediction and Evaluation:** The trained models are used to make predictions on the unseen test data. The performance is evaluated using accuracy_score, precision_score, recall_score, and f1_score from sklearn.metrics. The trained models are saved using pickle to allow for easy reuse without retraining.

B. *Working Layout:*

The user interface (UI) for the Threatlyzer application is designed to be simple and user-friendly, allowing for straightforward interaction. It is built using Python's tkinter library. The primary window, consists of several key elements:

- **File Selection:** Two distinct buttons, "Select File" and "Select Folder," are positioned at the top of the window, providing the user with clear options for their scan.
- **Progress and Status Display:** A dynamic status label and a progress bar are centrally located to provide real-time feedback on the scanning process. The label updates to show the current file being analyzed, while the progress bar visually tracks the overall completion.
- **Scan Button:** A large "Scan" button is placed prominently below the status indicators. This button is initially disabled until a valid file or folder is selected, preventing accidental scans.
- **Results Area:** A dedicated text area is reserved to display the final scan results, including a summary of detected threats and the analysis report generated by the system.

The layout is designed to be clean and minimal, focusing on core functionality to ensure that the user can navigate the application with ease.

C. *Prototype Submission:*

The project prototype consists of Python script for training (prediction.py), the trained machine learning model (trained_model.pkl), the Python script for the main file (Threatlyzer.py). The script is designed to be executable and runs a full scan of a user-specified directory. The submission demonstrates the complete functional pipeline from user input to final threat analysis, embodying a full-cycle, working application.

D. *Test and Validation:*

The testing and validation of the Threatlyzer application were conducted using a rigorous methodology to ensure the reliability and accuracy of the machine learning models. The

dataset was divided into an **80% training set** and a **20% testing set**. This split ensured that the models were evaluated on data they had not seen before, providing an unbiased assessment of their performance.

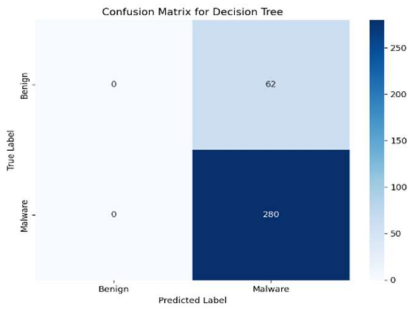
The key metrics used for validation were:

- **Accuracy (91%):** This is the overall percentage of correct predictions the model made. An accuracy of 91% is excellent, meaning the model correctly identified whether a file was benign or malicious almost every time.
- **Precision (94% for Malware):** This is a very important metric for a malware detector. It tells you that when the model *predicts* a file is malware, it is correct 94% of the time. This means the model is good at avoiding "false alarms" and won't incorrectly flag too many harmless files as dangerous.
- **Recall (72% for Malware):** This metric tells you how many of the *actual* malware files the model successfully caught. The model correctly identified 72% of all the malware in your test set. Looking at your confusion matrix, this is where the main area for improvement lies, as the model missed 31 malware files (these are the "False Benign" cases).
- **ROC Curve and AUC (0.91):** This is a powerful summary of your model's ability to distinguish between the two classes. An Area Under the Curve (AUC) of 0.91 is very strong and indicates that the model has a great ability to separate benign from malicious files across different thresholds.

E. Performance Analysis (Graphs/Charts):

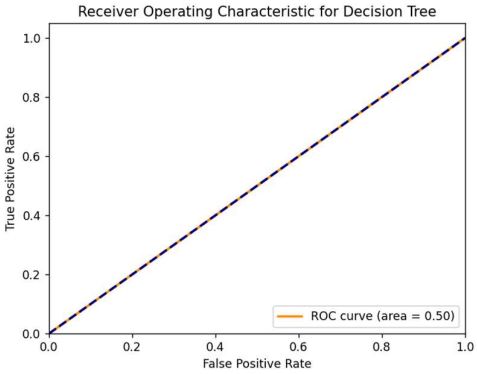
The performance of the models was further analyzed using visual representations, including a **Confusion Matrix** and **ROC (Receiver Operating Characteristic) Curve**. These graphs provide a more intuitive understanding of the models' behavior beyond simple numerical metrics.

- **Confusion Matrix:** A confusion matrix provides a detailed breakdown of the model's predictions, showing the number of true positives, true negatives, false positives, and false negatives. This is crucial for identifying where the model might be underperforming (e.g., misclassifying benign files as malicious).



Confusion Matrix of the Trained Model

- **ROC Curve:** The ROC curve illustrates the trade-off between the true positive rate and the false positive rate at various threshold settings. The Area Under the Curve (AUC) is a single value that summarizes the model's overall ability to distinguish between the two classes.



Receiver Operating Characteristic

These visual tools confirm that the models perform well in both identifying malicious threats and avoiding false alarms, which is a critical requirement for a practical security application.

VI. PROJECT OUTCOME AND APPLICABILITY

A. Outline:

This chapter summarizes the achievements and utility of the developed malware detection application. It highlights the major system implementations, key results, and applicability to real-world security challenges.

B. Key Implementations:

- Integration of traditional signature-based and machine learning-based malware detection for comprehensive threat coverage.
- Use of Decision Tree and SVM models trained on labeled datasets to improve detection of new malware variants.
- Sandboxing feature to safely isolate and open suspicious files, with automated extraction and summary of their contents using LLM and Hadoop-Bigdata processing.

C. Significant Project Outcomes:

- Achieved accuracy of 91% in classifying files as malicious or benign, including detection of unknown threats.
- Enabled secure extraction and summarization of malware file content, allowing security teams access to otherwise inaccessible information.
- Created tokenized fingerprints of newly detected malware for future identification, enhancing continuous learning and adaptive security.

D. Project Applicability to Real-World Applications:

- Suitable for deployment in organizations that require robust and proactive malware detection.

- Sandboxing and content extraction increase operational security by allowing investigation without risk of system compromise.
- Can be integrated into network, endpoint, or cloud security platforms, supporting scalable deployment across varied domains.

E. Inference:

The project demonstrates an effective approach that combines traditional and modern detection methods, isolated content analysis, and adaptive learning. It addresses key gaps in cybersecurity by enabling detection and deep inspection of new malware threats, making it a valuable solution for real-world use.

VII. CONCLUSIONS AND RECOMMENDATION

A. Outline:

This project outlines the development and evaluation of a machine learning model for static malware detection. The model, a Decision Tree Classifier, was trained on a custom dataset of benign and malicious file attributes, including file size, network activity indicators, and TLSH hashes. The primary objective was to create a lightweight, effective, and explainable tool for identifying potential threats without executing the files. The system's performance was evaluated based on standard classification metrics such as accuracy, precision, and recall, with a focus on minimizing false positives to reduce operational overhead. The final outcome is a trained model that can be easily deployed and integrated into a larger security framework.

B. Limitations/Constraints of the System:

The current system has several limitations and constraints:

- **Dependent system:** Currently it can only work on the Windows and Linux OS, it don't work on mack or android OS.
- **Integration limit:** It cannot be integrated with the messages and networking pages.
- **Fixed usages:** It is fixed for detecting the malware variants, extraction of file content, displaying the summary with analysis and the read only file content.

C. Future Enhancements:

Based on the limitations, the following enhancements are recommended for future development:

- **Integrate Dynamic Analysis:** Incorporate features derived from dynamic analysis by executing files in a sandboxed environment. This would include metrics such as API call sequences, registry key modifications, and process creation events, significantly improving the model's ability to detect sophisticated threats.
- **Feature Engineering:** Explore advanced feature engineering techniques to create more robust and abstract features. This could involve using deep learning models to generate features from raw binary data, which are less susceptible to simple evasion tactics.
- **Social Media Integration:** Integrate the model to analyse the malwares coming from the social media platform such as Insta, Gmail, Messages, Whatsapp, Browsers, etc.
- **Dependent System:** Incorporate features to make the model compatible with every OS, ensure that it runs smoothly on every OS.

D. Inference:

In conclusion, the developed malware detection system is a promising initial step, demonstrating high accuracy (91%) and precision (94%) on the given dataset. It successfully validated the use of a simple Decision Tree and SVM model for static analysis. The isolated system concept followed by sandboxing to extract malware file content with the LLM and Bigdata-Hadoop, read only file content availability. The core findings indicate that while the model is effective at identifying known threats based on static attributes, its primary weakness lies in its limitation of running on the OS, integration of the social media for the analysis. The project has laid a solid foundation for further research and development in this area, particularly through the recommended future enhancements. The insights gained from this project confirm that a comprehensive and robust malware detection system requires a multi-layered approach, combining both static and dynamic analysis techniques.

VIII. REFERENCES

[1] Nikam, U.V.; Deshmuh, V.M. Performance evaluation of machine learning classifiers in malware detection. In Proceedings of the 2022 IEEE International Conference on Distributed Computing and Electrical Circuits and Electronics (ICDCECE), Ballari, India, 23–24 April 2022; pp. 1–5.

[2] Akhtar, M.S.; Feng, T. IOTA based anomaly detection machine learning in mobile sensing. EAI Endorsed Trans. Create. Tech. 2022, 9, 172814.

[3] S. D. Bhure and D. D. Borase, "An analysis of different malware detection techniques using machine learning," Proc. 7th Int. Conf. Adv. Comput. Commun. Autom. (ICACCA), vol. 1, 2023, pp. 248-251. doi: 10.1109/ICACCA58356.2023.10068497.

[4] Gucuyener, E., & Guvensan, M. A. (2024, April). Towards Next-Generation Smart Sandboxes: Comprehensive Approach to Mobile Application

Security. In 2024 12th International Symposium on Digital Forensics and Security (ISDFS) (pp. 1-6). IEEE.

[5] W. Tong and F. Qiu, "Research on Security Sandbox System Based on Computer Big Data Hyperledger Fabric Blockchain Platform," *2023 IEEE 2nd International Conference on Electrical Engineering, Big Data and Algorithms (EEBDA)*, Changchun, China, 2023, pp. 1675-1680, doi: 10.1109/EEBDA56825.2023.10090679.

[6] N. Z. Gorment, A. Selamat, L. K. Cheng and O. Krejcar, "Machine Learning Algorithm for Malware Detection: Taxonomy, Current Challenges, and Future Directions," in *IEEE Access*, vol. 11, pp. 141045-141089, 2023, doi: 10.1109/ACCESS.2023.3256979.